data.table_note–introduction to data.table

王泽贤

2016 年 *12* 月 *7* 日

用 data.table 进行数据分析

诸如取子集、分组、更新、合并等数据操作本质上是联系在一起的。如果能够将这些相关操作联系在一起那么将可以：

- 统一一套简洁的关于所有数据操作的语法
- 减轻分析过程中由于需要了解不同作用函数的用法的认识负担
- 统一在一套系统中可以自动优化操作过程的内部计算，更加高效、快速、节省内存

简而言之，如果你希望能够极大地减少编程和计算的时间，那么这个包非常适合你。data.table的哲学就是希望如此。

数据介绍与读取

在本介绍中，我们采用 NYC-flights14数据集。这个数据集来源于 Bureau of Transporation Statistics，包括 2014 年 1 月到 10 月，从 New York City机场起飞的所有飞机的准点情况数据。

data.table中提供了数据读取函数 fread, 类似于 read.table但更快，要求数据的每一列的行数相同。可以直接读取 csv, 甚至是网页连接的数据。也可以在读取的时候直接选择所需的行和列进行读取。更多例子和说明请使用 help(fread)查询。

```
library(data.table)
flights <- fread("flights14.csv")
flights
```

```
##        year month day dep_time dep_delay arr_time arr_delay cancelled
##     1: 2014     1   1      914        14     1238        13         0
##     2: 2014     1   1     1157        -3     1523        13         0
##     3: 2014     1   1     1902         2     2224         9         0
##     4: 2014     1   1      722        -8     1014       -26         0
##     5: 2014     1   1     1347         2     1706         1         0
##    ---
## 253312: 2014    10  31     1459         1     1747       -30         0
## 253313: 2014    10  31      854        -5     1147       -14         0
## 253314: 2014    10  31     1102        -8     1311        16         0
## 253315: 2014    10  31     1106        -4     1325        15         0
## 253316: 2014    10  31      824        -5     1045         1         0
##        carrier tailnum flight origin dest air_time distance hour min
##     1:      AA  N338AA      1    JFK  LAX      359     2475    9  14
##     2:      AA  N335AA      3    JFK  LAX      363     2475   11  57
##     3:      AA  N327AA     21    JFK  LAX      351     2475   19   2
##     4:      AA  N3EHAA     29    LGA  PBI      157     1035    7  22
##     5:      AA  N319AA    117    JFK  LAX      350     2475   13  47
##    ---
## 253312:      UA  N23708   1744    LGA  IAH      201     1416   14  59
## 253313:      UA  N33132   1758    EWR  IAH      189     1400    8  54
## 253314:      MQ  N827MQ   3591    LGA  RDU       83      431   11   2
## 253315:      MQ  N511MQ   3592    LGA  DTW       75      502   11   6
## 253316:      MQ  N813MQ   3599    LGA  SDF      110      659    8  24
```

```
dim(flights)
```

[1] 253316   17

data.table 用法介绍

本节内容：

1. 选择行、选择和计算列
2. 分组汇总

1. 基础

a) 什么是 data.table

data.table是一个加强版的 data.frame。在数据介绍与读取小节中，我们已经通过 fread()创造了一个 data.table。除此之外，我们也可以直接用 data.table()函数直接创造一个 data.table

```
DT = data.table(ID = c("b","b","b","a","a","c"), a = 1:6, b = 7:12, c = 13:18)
DT
```

ID a b c
1: b 1 7 13
2: b 2 8 14
3: b 3 9 15
4: a 4 10 16
5: a 5 11 17
6: c 6 12 18

```
class(DT$ID)
```

[1] "character"

不同与 data.frame的默认形式，data.table默认不转化字符为因子，可以避免一些不必要的麻烦。需要的时候可以通过 factor()转化所需的列成为因子。

也可以通过 as.data.table()将已有数据对象转化为 data.table。

另外通过 datatable.print.nrows (100)，可以制定在命令行中输入一个数据集名字后，显示出来的行数 (比如 100)，而不是一次性显示所有内容。

b) data.table 加强在什么地方？

data.table引入了 SQL语言中的一些概念，并加以简化，对应如下：

DT[i,j,by]
R: i | j |by
SQL: where select | update group |by

如果你使用过 SQL 语言，那么将能够更快上手。

DT[i,j,by] 翻译成人类语言就是：对于 DT 这个数据集，取出其中满足 i条件的行，然后根据 j的要求对各个变量（列）计算，并且按照 by的要求进行分组。

下文中我们称 i,j,by为参数

c) 根据 i 参数选择行

- 选出航班中，origin变量是 “JFK”，且 month变量是六月的数据

```
ans <- flights[origin == "JFK" & month == 6L]
head(ans)
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     6   1      851        -9     1205        -5         0      AA
## 2: 2014     6   1     1220       -10     1522       -13         0      AA
## 3: 2014     6   1      718        18     1014        -1         0      AA
## 4: 2014     6   1     1024        -6     1314       -16         0      AA
## 5: 2014     6   1     1841        -4     2125       -45         0      AA
## 6: 2014     6   1     1454        -6     1757       -23         0      AA
##    tailnum flight origin dest air_time distance hour min
## 1:  N787AA      1    JFK  LAX      324     2475    8  51
## 2:  N795AA      3    JFK  LAX      329     2475   12  20
## 3:  N784AA      9    JFK  LAX      326     2475    7  18
## 4:  N791AA     19    JFK  LAX      320     2475   10  24
## 5:  N790AA     21    JFK  LAX      326     2475   18  41
## 6:  N785AA    117    JFK  LAX      329     2475   14  54
```

需要注意的是多个条件同时满足要用 &连接，如果是满足其中一个即可那要用 |连接。==表示相等条件。

在 data.table中，引用列也可以使用传统的 data.frame方式：flights$month 来完成

在 data.table中，直接用 flights[条件]即可筛选，当然用 data.frame方式：flights[条件,]也是可以的。

- 选出最开头的两行

```
ans <- flights[1:2]
ans
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     1   1      914        14     1238        13         0      AA
## 2: 2014     1   1     1157        -3     1523        13         0      AA
##    tailnum flight origin dest air_time distance hour min
## 1:  N338AA      1    JFK  LAX      359     2475    9  14
## 2:  N335AA      3    JFK  LAX      363     2475   11  57
```

可以直接通过行号向量来指定抽取的行，同样的，可以省略逗号。

- 对所有数据按照 origin列升序排列，之后再按照 dest列降序 (加上负号) 排列，order中可以包含一个或以上的排序条件。

```
ans <- flights[order(origin,-dest)]
head(ans)
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     1   5      836         6     1151        49         0      EV
## 2: 2014     1   6      833         7     1111        13         0      EV
## 3: 2014     1   7      811        -6     1035       -13         0      EV
## 4: 2014     1   8      810        -7     1036       -12         0      EV
## 5: 2014     1   9      833        16     1055         7         0      EV
## 6: 2014     1  13      923        66     1154        66         0      EV
##    tailnum flight origin dest air_time distance hour min
## 1:  N12175   4419    EWR  XNA      195     1131    8  36
## 2:  N24128   4419    EWR  XNA      190     1131    8  33
## 3:  N12142   4419    EWR  XNA      179     1131    8  11
## 4:  N11193   4419    EWR  XNA      184     1131    8  10
## 5:  N14198   4419    EWR  XNA      181     1131    8  33
```

6: N12157 4419 EWR XNA 188 1131 9 23

data.table中提供了高速版本的 order函数，通过这个函数可以使排序速度提高 15 倍以上。在大数据集中提升更加明显。

d) 通过 j 参数选择列

- 选出 arr_delay列，返回向量

```
ans <- flights[, arr_delay]
head(ans)
```

[1] 13 13 9 -26 1 0

对于 j 参数，由于排在 i 参数之后，为了避免混淆所以每次使用都需要在前面加上 i 参数的条件和逗号，如果没有 i 参数条件，那么直接加个逗号即可。

注意此时返回的是一个向量。如果我们希望返回的是 data.table则:

- 选出 arr_delay列，返回 data.table

```
ans <- flights[, list(arr_delay)]
head(ans)
```

```
##    arr_delay
## 1:        13
## 2:        13
## 3:         9
## 4:       -26
## 5:         1
## 6:         0
```

除此之外还有一个更加简略的写法

```
ans <- flights[, .(arr_delay)]
head(ans)
```

```
##    arr_delay
## 1:        13
## 2:        13
## 3:         9
## 4:       -26
## 5:         1
## 6:         0
```

同样可以返回一个 data.table。后文中我们都用这个简略写法。

j 参数配合.()或者 list()还有更多灵活的用法比如：

- 选择多列，如 arr_delay和 dep_delay

```
ans <- flights[, .(arr_delay, dep_delay)]
ans
```

```
##         arr_delay dep_delay
##      1:        13        14
##      2:        13        -3
##      3:         9         2
##      4:       -26        -8
##      5:         1         2
##     ---
## 253312:       -30         1
```

```
## 253313:      -14       -5
## 253314:       16       -8
## 253315:       15       -4
## 253316:        1       -5
```

- 选择多列并重命名，如将 arr_delay和 dep_delay重命名为 delay_arr和 delay_dep

```
ans <- flights[, .(delay_arr = arr_delay, delay_dep = dep_delay)]
head(ans)
```

```
##    delay_arr delay_dep
## 1:        13        14
## 2:        13        -3
## 3:         9         2
## 4:       -26        -8
## 5:         1         2
## 6:         0         4
```

e) 通过 j 参数对列进行计算

- 有多少航班的总延迟时间小于 0？（即提前到达）

```
ans <- flights[, sum((arr_delay + dep_delay)<0)]
ans
```

```
## [1] 141814
```

翻译：对 flights 数据集，选择所有行，计算判断每行 (arr_delay + dep_delay)<0 是否为 TRUE，并求和计算总数量，并返回结果。

简而言之 j 参数中的变量名可以直接当成普通的一个变量来应用，并进行计算。

f) 选择 i 条件的行，并进行 j 要求的计算

- 计算所有 origin 变量为 “JFK” 的航班，在 6 月的的平均到达和出发延误时间

```
ans <- flights[origin == "JFK" & month == 6L,
               .(m_arr = mean(arr_delay),m_dep = mean(dep_delay))]
ans
```

```
##      m_arr    m_dep
## 1: 5.839349 9.807884
```

- 6 月从 “JFK” 起飞的航班有多少？

```
ans <- flights[origin == "JFK" & month ==6L, length(dest)]
ans
```

```
## [1] 8422
```

上述做法是通过选择满足 i 参数的行，然后计算其中任意一个变量（这边选了 dest）的长度，从而获得满足条件的行数。

```
ans <- flights[origin == "JFK" & month == 6L, .N]
ans
```

```
## [1] 8422
```

第二种做法通过.N替代 length(dest)来计算个数，.N是 data.table中用于返回行数的特殊符号。

当然也可以使用 nrow(flights[origin == "JFK" & month == 6L])来实现，但是这样做的速度和内存开销都比直接加入.N符号效果差。

g) 通过.SD 和.SDcols 在 j 中灵活选取多个列

前面的方法中，如果要计算多个列的均值，我们需要 flights[, .(m_arr = mean(arr_delay),m_dep = mean(dep_delay))], 然而如果有成千上万列，一个一个写进.()不太现实。data.table中提供了.SD符号来解决这个问题。

.SD符号表示 “所有变量 “，换言之，使用.SD 就相当于把所有变量的名字写进一个向量中。

前面我们已经生成了一个简单的数据集 DT, 为了方便演示，我们使用这个数据集进行说明

```
DT
```

```
##    ID a  b  c
## 1:  b 1  7 13
## 2:  b 2  8 14
## 3:  b 3  9 15
## 4:  a 4 10 16
## 5:  a 5 11 17
## 6:  c 6 12 18
```

显然，ID变量是无法计算均值的，它是文本向量。直接使用.SD把所有变量扔进去算均值肯定要出错。data.table又提供了.SDcols来解决这个问题。

.SDcols符号表示.SD符号中要包含的变量名字，例如指定.SDcols = c(”arr_delay”,”dep_delay”)就可以让.SD只引用 2 个变量。

也可以用.SDcols = -c(”arr_delay”,”dep_delay”)，通过假上一个负号（感叹号! 也行），来指定.SD中取所有变量，但是不包含这 2 个变量

一个具体例子：

```
DT[, lapply(.SD, mean), .SDcols = -c(”ID”)]
```

```
##     a   b    c
## 1: 3.5 9.5 15.5
```

其中的 lapply(.SD, mean)表示把.SD 中的所有东西，一个一个地放入 mean()函数，并返回结果。你可以指定其他函数名进行计算

2. 分组汇总

a) 通过 by 函数分组

- 计算每个 origin变量类别的航班数

```
ans <- flights[, .(.N), by = .(origin)]
ans
```

```
##    origin     N
## 1:    JFK 81483
## 2:    LGA 84433
## 3:    EWR 87400
```

也可以通过 flights[, .N, by = ”origin”]来完成。

- 计算每个 origin变量类别中 carrier变量为 “AA” 的航班数

```
ans <- flights[carrier == ”AA”, .N, by = origin]
ans
```

```
##    origin     N
## 1:    JFK 11923
## 2:    LGA 11730
## 3:    EWR  2649
```

- 交叉分组：计算每组 origin和 dest变量中，carrier变量为 “AA” 的航班数

```
ans <- flights[carrier == ”AA”, .N, by = .(origin,dest)]
ans
```

```
##    origin dest    N
## 1:    JFK  LAX 3387
## 2:    LGA  PBI  245
## 3:    EWR  LAX   62
## 4:    JFK  MIA 1876
## 5:    JFK  SEA  298
## 6:    EWR  MIA  848
## 7:    JFK  SFO 1312
## 8:    JFK  BOS 1173
## 9:    JFK  ORD  432
## 10:    JFK  IAH    7
## 11:    JFK  AUS  297
## 12:    EWR  DFW 1618
## 13:    LGA  ORD 4366
## 14:    JFK  STT  229
## 15:    JFK  SJU  690
## 16:    LGA  MIA 3334
## 17:    LGA  DFW 3785
## 18:    JFK  LAS  595
## 19:    JFK  MCO  597
## 20:    JFK  EGE   85
## 21:    JFK  DFW  474
## 22:    JFK  SAN  299
## 23:    JFK  DCA  172
## 24:    EWR  PHX  121
##    origin dest    N
```

也可以用 flights[carrier == ”AA”, .N, by = c(”origin”, ”dest”)]来实现。

- 更复杂的情况，想想这是什么意思？

```
ans <- flights[carrier == ”AA”,
  .(mean(arr_delay), mean(dep_delay)),
  by = .(origin, dest, month)]
ans
```

```
##    origin dest month        V1        V2
##  1:    JFK  LAX     1  6.590361 14.2289157
##  2:    LGA  PBI     1 -7.758621  0.3103448
##  3:    EWR  LAX     1  1.366667  7.5000000
##  4:    JFK  MIA     1 15.720670 18.7430168
##  5:    JFK  SEA     1 14.357143 30.7500000
##  ---
## 196:    LGA  MIA    10 -6.251799 -1.4208633
## 197:    JFK  MIA    10 -1.880184  6.6774194
## 198:    EWR  PHX    10 -3.032258 -4.2903226
## 199:    JFK  MCO    10 -10.048387 -1.6129032
## 200:    JFK  DCA    10  16.483871 15.5161290
```

我们可以看到 by 函数返回的时候顺序是按原始数据的顺序返回的。

b) keyby

如果我们希望按照分组变量的内容来排序：

```
ans <- flights[carrier == "AA",
   .(mean(arr_delay), mean(dep_delay)),
   keyby = .(origin, dest, month)]
ans
```

```
##     origin dest month        V1        V2
##  1:    EWR  DFW     1  6.427673 10.0125786
##  2:    EWR  DFW     2 10.536765 11.3455882
##  3:    EWR  DFW     3 12.865031  8.0797546
##  4:    EWR  DFW     4 17.792683 12.9207317
##  5:    EWR  DFW     5 18.487805 18.6829268
## ---
## 196:    LGA  PBI     1 -7.758621  0.3103448
## 197:    LGA  PBI     2 -7.865385  2.4038462
## 198:    LGA  PBI     3 -5.754098  3.0327869
## 199:    LGA  PBI     4 -13.966667 -4.7333333
## 200:    LGA  PBI     5 -10.357143 -6.8571429
```

by 改成 keyby 即可

keyby 在内部是首先用 by 分组计算后，再进行的位置调整。

c) Chaining 连锁操作

- 考虑之前的交叉分组：计算每组 origin和 dest变量中，carrier变量为 “AA” 的航班数

```
ans <- flights[carrier == "AA", .N, by = .(origin,dest)]
ans
```

```
##     origin dest    N
##  1:    JFK  LAX 3387
##  2:    LGA  PBI  245
##  3:    EWR  LAX   62
##  4:    JFK  MIA 1876
##  5:    JFK  SEA  298
##  6:    EWR  MIA  848
##  7:    JFK  SFO 1312
##  8:    JFK  BOS 1173
##  9:    JFK  ORD  432
## 10:    JFK  IAH    7
## 11:    JFK  AUS  297
## 12:    EWR  DFW 1618
## 13:    LGA  ORD 4366
## 14:    JFK  STT  229
## 15:    JFK  SJU  690
## 16:    LGA  MIA 3334
## 17:    LGA  DFW 3785
## 18:    JFK  LAS  595
## 19:    JFK  MCO  597
## 20:    JFK  EGE   85
## 21:    JFK  DFW  474
## 22:    JFK  SAN  299
## 23:    JFK  DCA  172
## 24:    EWR  PHX  121
##     origin dest    N
```

现在我们希望对得到的结果再根据某个变量排序，比如根据 origin升序，dest降序，则可以在结果后加入[order(origin, -dest)]进行连锁操作

```
ans <- ans[order(origin, -dest)]
head(ans)
```

```
##    origin dest    N
## 1:    EWR  PHX  121
## 2:    EWR  MIA  848
## 3:    EWR  LAX   62
## 4:    EWR  DFW 1618
## 5:    JFK  STT  229
## 6:    JFK  SJU  690
```

既然是叫连锁操作，自然我们可以连续写完一组操作

```
ans <- flights[carrier == "AA", .N, by = .(origin, dest)][order(origin, -dest)]
head(ans)
```

```
##    origin dest    N
## 1:    EWR  PHX  121
## 2:    EWR  MIA  848
## 3:    EWR  LAX   62
## 4:    EWR  DFW 1618
## 5:    JFK  STT  229
## 6:    JFK  SJU  690
```

加入更多的 [...]可以进行多步连锁操作

d) by 中的表达式

事实上 by 函数不仅仅可以用来选择分组变量，如果在其中使用表达式也可以做到用计算后变量分组的效果

```
ans <- flights[, .N, .(dep_delay>0, arr_delay>0)]
ans
```

```
##    dep_delay arr_delay     N
## 1:     TRUE      TRUE  72836
## 2:    FALSE      TRUE  34583
## 3:    FALSE     FALSE 119304
## 4:     TRUE     FALSE  26593
```

当然也可以只对一个变量计算，另一个变量直接分组

```
ans <- flights[, .N, .(origin, arr_delay>0)]
ans
```

```
##    origin arr_delay     N
## 1:    JFK      TRUE 34636
## 2:    LGA     FALSE 49703
## 3:    EWR     FALSE 49347
## 4:    JFK     FALSE 46847
## 5:    EWR      TRUE 38053
## 6:    LGA      TRUE 34730
```

e) 组合使用

```
flights[carrier == "AA",
  lapply(.SD, mean),
```

```
  by = .(origin, dest, month),
  .SDcols = c("arr_delay", "dep_delay")]
```

```
##      origin dest month  arr_delay  dep_delay
##   1:    JFK  LAX     1   6.590361 14.2289157
##   2:    LGA  PBI     1  -7.758621  0.3103448
##   3:    EWR  LAX     1   1.366667  7.5000000
##   4:    JFK  MIA     1  15.720670 18.7430168
##   5:    JFK  SEA     1  14.357143 30.7500000
##  ---
## 196:    LGA  MIA    10  -6.251799 -1.4208633
## 197:    JFK  MIA    10  -1.880184  6.6774194
## 198:    EWR  PHX    10  -3.032258 -4.2903226
## 199:    JFK  MCO    10 -10.048387 -1.6129032
## 200:    JFK  DCA    10  16.483871 15.5161290
```

3. 简单数据塑型

回顾下开头创建的 DT

```
DT
```

```
##    ID a  b  c
## 1:  b 1  7 13
## 2:  b 2  8 14
## 3:  b 3  9 15
## 4:  a 4 10 16
## 5:  a 5 11 17
## 6:  c 6 12 18
```

- 把每种 ID 的 b 数据接到 a 数据下

```
DT[, .(val = c(a,b)), by = ID]
```

```
##     ID val
##  1:  b   1
##  2:  b   2
##  3:  b   3
##  4:  b   7
##  5:  b   8
##  6:  b   9
##  7:  a   4
##  8:  a   5
##  9:  a  10
## 10:  a  11
## 11:  c   6
## 12:  c  12
```

- 把每种 ID 的 b 数据和 a 数据连接并返回 list

```
DT[, .(val = list(c(a,b))), by = ID]
```

```
##    ID        val
## 1:  b 1,2,3,7,8,9
## 2:  a  4, 5,10,11
## 3:  c       6,12
```

对比：

```
DT[, print(c(a,b)), by = ID]
```

```
## [1] 1 2 3 7 8 9
## [1]  4  5 10 11
## [1]  6 12
```

```
## Empty data.table (0 rows) of 1 col: ID
```

```
DT[, print(list(c(a,b))), by = ID]
```

```
## [[1]]
## [1] 1 2 3 7 8 9
##
## [[1]]
## [1]  4  5 10 11
##
## [[1]]
## [1]  6 12
```

```
## Empty data.table (0 rows) of 1 col: ID
```

4. 总结

基本操作形式：DT[i, j, by]

Using i:

- We can subset rows similar to a data.frame - except you don't have to use DT$ repetitively since columns within the frame of a data.table are seen as if they are variables.
- We can also sort a data.table using order(), which internally uses data.table's fast order for performance.

We can do much more in i by keying a data.table, which allows blazing fast subsets and joins. We will see this in the "Keys and fast binary search based subsets" and "Joins and rolling joins" vignette.

Using j:

- Select columns the data.table way: DT[, .(colA, colB)].
- Select columns the data.frame way: DT[, c("colA", "colB"), with = FALSE].
- Compute on columns: DT[, .(sum(colA), mean(colB))].
- Provide names if necessary: DT[, .(sA =sum(colA), mB = mean(colB))].
- Combine with i: DT[colA > value, sum(colB)].

Using by:

- Using by, we can group by columns by specifying a list of columns or a character vector of column names or even expressions. The flexibility of j, combined with by and i makes for a very powerful syntax.
- by can handle multiple columns and also expressions.
- We can keyby grouping columns to automatically sort the grouped result.
- We can use .SD and .SDcols in j to operate on multiple columns using already familiar base functions. Here are some examples:

1. DT[, lapply(.SD, fun), by = …, .SDcols = …] - applies fun to all columns specified in .SDcols while grouping by the columns specified in by.
2. DT[, head(.SD, 2), by = …] - return the first two rows for each group.
3. DT[col > val, head(.SD, 1), by = …] - combine i along with j and by.